IBM Technical Newsletter

11

File Number	S360/25
Re: Form No.	C28-6583-0
This Newsletter No.	N28-2108
Date	July 16, 1965
Previous Newsletter Nos.	None

IBM Operating System/360 Basic Programming Support
FORTRAN IV, 360P-FO-031
Programmer's Guide

This technical newsletter amends the publication IBM System/360 Basic Programming Support FORTRAN IV, 360P-FO-031 Programmer's Guide, Form C28-6583-0. The attached pages, 7 through 12, 33 through 42, 45, 46, 49 through 52, 55, 56, 69, 70, 77, 78, 81, 82, should be inserted in place of the previous version of these pages. Text changes are indicated by a vertical line to the left of the affected text; figure changes are indicated by a bullet (•) to the left of the figure caption.

Summary of Amendments

Internal modification of the library subprogram names to improve loading efficiency has been described.

Figures 2, 28, 32, 33, and 61, and the examples on pages 34 and 35 have been replaced. Figure 66.1 has been added.

A message requiring operator intervention will be printed when an abnormal end-of-job occurs due to interruptions.

The description of error code diagnostic 200 has been changed; error code diagnostics 253 and 263 have been amended.

In Appendix D, the descriptions of the MOD, AMOD, and DMOD subprograms have been amended.

Note

Please file this cover letter at the back of the publication. Cover letters provide a quick reference to changes, and a means of checking receipt of all amendments.

International Business Machines Corp., Programming Systems Publications, P. O. Box 390, Poughkeepsie, N. Y.

PRINTED IN U.S.A.

This section provides instructions for compiling and executing programs written in IBM System/360 Basic Programming Support FORTRAN IV Language.

The Basic Programming Support FORTRAN compiler accepts as input one or more source or object programs (or subprograms) in many combinations, called a FORTRAN "job," along with any associated input data.

In preparing a FORTRAN job, the user provides control information in control statements to tell the compiler how to handle a given type of job. To reduce the possibility of errors in using control statements, the compiler provides "standard options;" that is, in the absence of control information, the compiler makes assumptions on how to handle a given job.

The control statements, their placement within an input deck, diagnostic messages produced at source and object time, machine requirements, and program size considerations are described in the following text.

CONTROL STATEMENTS

Control statements are nonexecutable statements that inform the FORTRAN compiler about the nature of a job. There are two categories of control statements: compilation and execution statements; and FORTRAN system tape maintenance statements. The latter are described in the section "FORTRAN System Maintenance."

The compilation and execution control statements specify the following type of information:

1. The name of the job and/or the name of each individual program within it.
2. Whether the job consists of one program, or a combination of source and object programs.
3. Whether the programs within a job are to be:
   a. Compiled only,
   b. Compiled and executed, or
   c. Executed only.
4. The type of output desired, such as program listing, object program card deck, or storage layout map.
5. The end of a particular job or the beginning of input data, if any.
6. The correspondence between data set reference numbers used in a program and the actual input/output (I/O) device addresses for a particular job.

Five control statements are used to specify the preceding information. They are: /JOB, /FTC, /DATA, /SET, and /LOAD.

With the exception of the /DATA control statement, all control statements are optional and have optional parameter lists. The absence of a particular parameter within a control statement, or in fact the statement itself, implies that a predefined specification (known as a standard option) is to be assumed by the compiler. Each control statement, along with its parameters, is described in the following text. Examples of control statements are in card format showing the column numbers in which the parameter list (if any) must begin.

/JOB CONTROL STATEMENT

The /JOB control statement is the first statement of a job. It has two purposes: (1) specifies whether the job consists of one or more compilations and (2) informs the FORTRAN compiler whether the job is to be compiled and executed or compiled only. (If the job consists solely of object programs, it may be executed by a /LOAD control statement; see the section, "/LOAD Control Statement.")

Associated with the /JOB control statement is a parameter list composed of two option classes:

1. SINGLE or MULTIPLE.
2. GO, GOGO, or NOGO.

SINGLE or MULTIPLE Option

The parameters SINGLE and MULTIPLE inform the FORTRAN compiler about the number of compilations to be performed for a certain job (i.e., one or several).

The parameter SINGLE specifies that the job consists of a single compilation of a source program or source subprogram. The job to which this parameter applies should not comprise more than one source program because only one compilation process is specified. However, as many object programs as desired may be included in the job.

The parameter MULTIPLE specifies that the job consists of several compilations, depending on the number of source programs and source subprograms in the job. The job itself may comprise as many object programs as desired because the parameter applies only to the number of compilations in a job; not the total number of programs in a job.

Embedded blanks are not permitted within the parameters. If neither SINGLE nor MULTIPLE is specified, the standard option is MULTIPLE.

GO, GOGO, or NOGO Options

The parameter GO specifies that the entire job is to be compiled and executed. If there are no serious source program errors, each program in the job (i.e., the compiled source programs and any object programs) is written on a tape (called a GO tape). When the entire job has been compiled and written on the GO tape, the tape is loaded into storage and execution commences. If, however, there are serious source program errors, execution is not attempted (see the section, "Diagnostic Messages").

The parameter GOGO also specifies that the job is to be compiled and executed. However, unlike the GO option, a GO tape is always produced and execution of the job is attempted regardless of any errors existing in the job. Note, however, that these errors may cause premature termination of the job.

The parameter NOGO specifies that the job is to be compiled only; no execution occurs and no GO tape is produced.

Embedded blanks are not permitted within the parameters. If neither GO, GOGO, nor NOGO is specified, the standard option is GO.

Format of /JOB Statement

Figure 1 shows the format of the /JOB statement.

```
Column 1   Column 10
↑          ↑
           SINGLE        GO
/JOB                  ,  NOGO
           MULTIPLE      GOGO
```

Figure 1. Format of /JOB Statement

The following are considerations in using the /JOB statement:

1. /JOB must start in column 1.

2. The option classes may appear in any order, the first starting in column 10.

   <u>Examples</u>:

   ```
   /JOB     SINGLE,NOGO
   /JOB     GO
   /JOB     GOGO,MULTIPLE
   /JOB
   ```

3. Each parameter must be separated from the other only by a comma. The first blank encountered after column 10 results in whatever follows being regarded as comments. This comments field may be used for specifying the name of the job. The entire /JOB statement is always printed on-line.

   <u>Examples</u>:

   ```
   /JOB     SINGLE,GO PROBLEM1
   /JOB      PROBLEM2
   ```

4. If two parameters in the same option class appear in a /JOB statement, the one appearing last is used.

5. If a /JOB statement is omitted from the job, the standard options assumed are:

   ```
   /JOB     MULTIPLE,GO
   ```

6. Only one /JOB statement should be used for each job.

/FTC CONTROL STATEMENT

The /FTC control statement may precede each source program or source subprogram in a job. It has three purposes: (1) permits the user to uniquely name each program in his job, (2) requests optional output from a compilation such as object program on cards (deck), printout of source program, and a layout of storage reserved for program, and (3) allows the user to specify the size (in bytes) of main storage in which the job is compiled and/or executed.

Associated with the /FTC statement is a parameter list composed of five option classes:

1. NAME=xxxxxx, where xxxxxx is 1 through 6 characters, the first of which must be alphabetic.
2. DECK or NODECK
3. LIST or NOLIST
4. MAP or NOMAP
5. SIZE=xK, where x is in the range, $16 \leq x \leq 64$, and K represents 1024 bytes.

NAME=xxxxxx Option

The parameter NAME=xxxxxx specifies the name the user wishes to assign to the program or subprogram. If the job consists of more than one program, additional /FTC statements with different names may be used, one for each program. If this is done, each /FTC statement should precede each program, respectively. However, there is no necessity in specifying a name for each program in a job. If no specification is made, the assumed name for a main program is MAIN; for a subprogram the assumed name is the subprogram name in the header source statement (e.g., FUNCTION SUM(X,Y)).

Embedded blanks are not permitted within the parameter NAME=xxxxxx.

DECK or NODECK Option

The parameter DECK specifies that the compiled source program (i.e., the object program) is to be punched on cards; the parameter NODECK specifies no object program deck output.

If the job is GO or GOGO and the DECK option is specified, the object program deck is produced in addition to the GO tape containing the object programs. These object programs produced on the GO tape are loaded into storage for execution; the object deck is not used. It is merely additional output requested by the programmer.

If the job is NOGO, the compiled source program can be executed only if the DECK option is specified. In this case, however, execution of the object deck is considered to be a different job. If it is desired to compile and execute all in one job, the GO or GOGO option should be specified.

Embedded blanks are not permitted within the parameters. If neither DECK nor NODECK is specified, the standard option is NODECK.

LIST or NOLIST Option

The parameter LIST specifies that the source program is to be printed (listed) on the device associated with data set reference number 3 (usually a printer); the parameter NOLIST specifies no listing.

Embedded blanks are not permitted within the parameters. If neither LIST nor NOLIST is specified, the standard option is LIST.

An example of a LIST printout is shown in Figure 2. This printout is the source program listing of Sample Program 1, shown in Appendix D, of the publication IBM System/360: Basic Programming Support FORTRAN IV, Form C28-6504.

```
BPS FORTRAN IVD COMPILER  VERSION 1  LEVEL 0   JUNE 1965
/FTC     DECK
 BEGIN COMPILATION
           C      PRIME NUMBER PROBLEM
 S.0001       100 WRITE (6,8)
 S.0002         8 FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
                 119X,1H1/19X,1H2/19X,H3)
 S.0003       101 I=5
 S.0004         3 A=I
 S.0005       102 A=SQRT(A)
 S.0006       103 J=A
 S.0007       104 DO 1 K=3,J,2
 S.0008       105 L=I/K
 S.0009       106 IF(L*K-I) 1,2,4
 S.0010         1 CONTINUE
 S.0011       107 WRITE (6,5) I
 S.0012         5 FORMAT (I20)
 S.0013         2 I=I+2
 S.0014       108 IF(1000-I) 7,4,3
 S.0015         4 WRITE (6,9)
 S.0016         9 FORMAT (14H PROGRAM ERROR)
 S.0017         7 WRITE (6,6)
 S.0018         6 FORMAT (31H THIS IS THE END OF THE PROGRAM)
 S.0019           CALL DUMP(I,L)
 S.0020       109 STOP
 S.0021           END

END OF COMPILATION MAIN
```

Note: The numbers S.0001 through S.0021 represent sequential internal statement numbers which are assigned to each source statement by the compiler. Comments lines and continuation lines are not assigned internal statement numbers.

• Figure 2. Sample Program Listing

MAP or NOMAP Option

The parameter MAP specifies that a storage map of the source and object program is to be printed on-line; the parameter NOMAP specifies no printout.

A storage map at compilation time gives a complete listing of:

1. The relative addresses and names of all variables used in the source program, including FUNCTION names, and in-line subprogram names.
2. The relative addresses and names of all external references made by the program, including all subprograms except in-line subprograms.
3. All user-specified literal constants in a program along with their relative addresses (e.g., in the statement A=2.0; 2.0 is a user-specified literal constant).
4. All compiler-generated constants along with their relative addresses.
5. A branch list consisting of all referenced statement numbers (except FORMAT statement numbers) in a program along with their relative addresses.
6. The relative address of the entry point where execution is to begin.
7. The size in bytes of the program.
8. The size in bytes of the blank COMMON area.

A storage map at execution time gives a complete listing of the absolute addresses (in hexadecimal) at which each object program in a job is loaded.

Embedded blanks within the MAP or NOMAP parameters are not permitted. If neither option is specified, the standard option is MAP.

An example of a MAP printout is shown in Figure 3. This printout is the source program map of Sample Program 1, shown in Appendix D, of the publication IBM System/360: Basic Programming Support FORTRAN IV, Form C28-6504.

Form C28-6583-0
Page Revised 7/16/65
By TNL N28-2108

```
                   STORAGE MAP  VARIABLES (TAG C = COMMON, E = EQUIVALENCE)

NAME TAG     REL ADR     NAME TAG     REL ADR     NAME TAG     REL ADR     NAME TAG     REL ADR

I            000090      A            000094      J            000098      K            00009C
L            0000A0

                                    EXTERNAL REFERENCES

NAME         REL ADR     NAME         REL ADR     NAME         REL ADR     NAME         REL ADR
SQRT         0000A4      DUMP         0000A8

                                         CONSTANTS

NAME         REL ADR     NAME         REL ADR     NAME         REL ADR     NAME         REL ADR
00000005     0000C4      00000002     0000C8      000003E8     0000CC

                               IMPLIED EXTERNAL REFERENCES

NAME         REL ADR     NAME         REL ADR     NAME         REL ADR     NAME         REL ADR

IBCOM        000168
STATEMENT NO. REL ADR STATEMENT NO. REL ADR STATEMENT NO. REL ADR STATEMENT NO. REL ADR
  00100        00019E       00008     0000D0       00101     0001B4       00003     0001BC
  00102        0001DC       00103     0001EA       00104     000208       00105     000210
  00106        000220       00001     000238       00107     00024C       00005     00011C
  00002        000268       00108     000274       00004     000288       00009     000120
  00007        00029C       00006     000134       00109     0002BA
                     SIZE OF COMMON   00000
                     SIZE OF PROGRAM  00712
```

Note: The name and relative address of each statement number, constant, variable, and external reference is printed as shown. In addition, a tag of C or E is printed for each variable specifying whether it is in COMMON or EQUIVALENCE, respectively; otherwise, the tag field is left blank.

•Figure 3. Sample Program Storage MAP

SIZE=xK Option

The parameter SIZE=xK, where x is in the range, 16≤x≤64, specifies the size in bytes of usable main storage in which the source program is compiled. (K represents 1024 bytes.) The compilation speed of a job may be affected appreciably by the size of the machine. (In general, the larger the machine, the faster the compilation.)

If the specified SIZE option is greater than 64K (e.g., SIZE=256K), only the first 64K bytes of main storage are used for the compilation. However, all main storage (e.g., 256K) is used for the execution.

Embedded blanks are not permitted within the SIZE=xK parameter. If no SIZE option is specified, the standard option is SIZE=16K.

Format of /FTC Statement

Figure 4 shows the format of the /FTC statement.

```
Column 1  Column 10
↑         ↑
                          DECK      LIST      MAP
/FTC      NAME=xxxxxx  ,        ,         ,         ,  SIZE=xK
                          NODECK    NOLIST    NOMAP
```

Figure 4. Format of /FTC Statement

The following are considerations in using the /FTC statement:

1. /FTC must start in column 1.

2. The parameters may appear in any order, the first starting in column 10.

   Examples:

```
/FTC     NODECK,LIST,NOMAP,SIZE=32K
/FTC     NAME=PROB1,SIZE=64K,NODECK
/FTC
/FTC     LIST,MAP,NODECK,SIZE=16K
```

   Note that the third and fourth examples above are effectively the same statements (see item 6).

3. Each parameter must be separated from the next only by a comma. The first blank encountered after column 10 results in whatever follows being regarded as comments. This comments field may be used for any purpose. The entire /FTC statement is printed along with comments whether or not the LIST or NOLIST option is specified.

4. If two parameters in the same option class appear in a /FTC statement, the one appearing last is used.

5. If a /FTC statement precedes only the first program in a job, all subsequent programs in the job are subject to the same specifications stated or implied in that statement. However, if an additional /FTC statement is encountered prior to any other program in a job, that program and all remaining programs (until the next /FTC statement appears) are subject to its specifications. For example, the following statements:

```
/FTC     NODECK,NOMAP,SIZE=32K
   .
   .
   .
/FTC     NOLIST
   .
   .
   .
```

   results in the specification made in the first /FTC statement to hold for all programs up to the second /FTC statement. Thereafter, the second /FTC statement (along with its specified and standard options) is considered the specification for all remaining programs in the job.

6. At object time, the specifications in the last /FTC statement in the job take precedence. For example:

```
/FTC     SIZE=64K
   .
   .
   .
/FTC     NAME=PROB2
   .
   .
   .
```

   The second /FTC statement would cause the SIZE option to be reset at 16K (the standard option). At object time the entire job would be executed assuming 16K of storage.

7. If no /FTC statements appear in a job, the standard options assumed for the entire job are:

```
/FTC     NODECK,LIST,MAP,SIZE=16K
```

/DATA CONTROL STATEMENT

The /DATA control statement is used to terminate all jobs. It should appear as the first card after the last source (or object) program in a job or immediately preceding any object time data cards.

If no /DATA control statement is specified, a warning message is produced and any cards remaining in the input device (e.g., card reader) are read until an end-of-file condition is sensed. Any data cards for use during execution may have been read and will have been lost for that job; however, normal operating procedure resumes.

Embedded blanks are not permitted within the characters /DATA. The /DATA control statement has no parameters. Comments within the statement are not permitted.

Format of /DATA Statement

Figure 5 shows the format of the /DATA statement.

```
Column 1
↑
/DATA
```

Figure 5. Format of /DATA Statement

The following are considerations in using the /DATA statement:

```
Column 1         Column 6         Column 13          Column 21
↑                ↑                ↑                  ↑

  EDR            address          segment name       object coding
```

Figure 27. Format of 12-2-9 EDR Statement

The following are considerations in using the EDR statement:

1. 12-2-9 EDR must start in column 1.

2. The first blank encountered in or after column 21 results in whatever follows being regarded as comments.

3. As many 12-2-9 EDR statements as needed may be used to modify a segment. However, the last 12-2-9 EDR statement used in modifying a particular segment must be followed by a 12-2-9 END statement.

Example: (The 12-2-9 punch is not shown in examples.)

```
          .
          .
          .
EDR       002A02 /EDR      F01A,235A,B302
EDR       002A46 /EDR      F20B,9E03
EDR       002B04 /EDR      2113,57EA
END       END OF EDITOR EDIT PROCESS
          .
          .
          .
```

Explanation: The first 12-2-9 EDR statement causes the specified code to replace existing code in the editor from location 002A02 through location 002A07. The next 12-2-9 EDR statement causes the specified code to replace the code from location 002A46 through location 002A49. The next 12-2-9 EDR statement then causes the existing code in the editor from location 002B04 through 002B07 to be replaced by 211357EA. The 12-2-9 END statement signals the termination of editing for the editor.

/AFTER CONTROL STATEMENT

The /AFTER control statement is used to insert user-written subprograms in the system library. These user-written subprograms must be in object deck format (i.e., object coding) produced from either a Basic Programming Support assembler source program assembly or a FORTRAN source program compilation.

Associated with the /AFTER control statement are two parameter classes:

1. Library subprogram name after which the user-written subprogram(s) is inserted.

2. User-written subprogram names that are to be incorporated in the system library.

Library Subprogram Name

Associated with every library subprogram is a subprogram name and one or more entry names representing particular entry points in the subprogram. The entry points designate a particular portion of the subprogram that performs a certain function. Only entry points should be referred to in a FORTRAN source program. For example, associated with the library subprogram named SIN are two entry points: SIN and COS. Both of these entry points may be used in a FORTRAN source program to compute the sine and cosine of a given argument, respectively. However, only subprogram names may be used in an /AFTER statement to specify the library subprogram after which a user-written subprogram(s) is to be inserted.

In some instances, the subprogram name and entry point for a particular library subprogram are identical. For example, the name of the library subprogram AINT represents not only the subprogram name but also the entry point for that subprogram. The name AINT may be used in a source program to truncate the fractional portion of an argument. This name may also be used in a /AFTER statement to designate the subprogram after which a specified user-written subprogram(s) is inserted. Figure 28 shows the complete list of library subprogram names and associated entry points (shown indented) supplied by IBM, in the order in which they appear on the FORTRAN system tape.

The library subprogram name must be placed starting in column 10. Embedded blanks within the name are not permitted.

```
  FDXPD¹       DUMP          MAX1
  FDXPI¹        PDUMP         MIN1
  FIXPI¹       DLOG           AMAX1
  FRXPI¹        DLOG10        AMIN1
  FRXPR¹       DSQRT         DMAX1
  EXIT         DATAN          DMIN1
  IBERR        DTANH         FLOAT
  CGOTO        DCOS           DFLOAT
  ALOG          DSIN         IFIX
   ALOG10      DEXP           INT
  SQRT         MOD            IDINT
  ATAN         AMOD          AINT
  TANH          DMOD         SLITE
  EXP          MAX0           SLITET
  COS           MIN0         OVERFL
   SIN          AMAX0        DVCHK
                AMIN0
```

¹Used at object-time to perform exponential operations.

NOTE: All indented names represent entry points. All other names are subprogram names as well as entry points.

• Figure 28. Library Subprograms Supplied by IBM

User-Written Subprogram Names

The user-written subprogram names are the names of the subprograms in object deck format to be inserted in the library. The name of the subprogram is the name specified in the /FTC statement (if any) used in compiling the subprogram. If no /FTC statement was used, the name of the subprogram is the name specified in the header source statement (e.g., for the statement SUBROUTINE HOLD(X,Y), HOLD is considered the subprogram name).

If a user wants to replace a library subprogram with his own of the same name, he must make a first edit run to delete the original subprogram and then add the new subprogram in a second edit run.

The user-written subprogram names are placed starting in column 17, each separated by a comma. Embedded blanks are not permitted within the list of names.

Format of /AFTER Statement

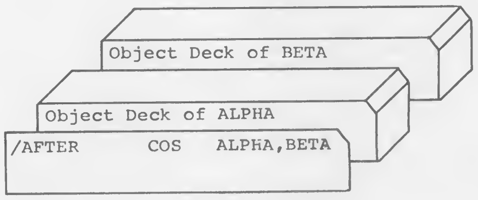
Figure 29 shows the format of the /AFTER statement.

```
Column 1  Column 10  Column 17
↑         ↑          ↑
/AFTER    name       sub₁,sub₂,...,subₙ
```

Figure 29. Format of /AFTER Statement

The following are considerations in using the /AFTER statement:

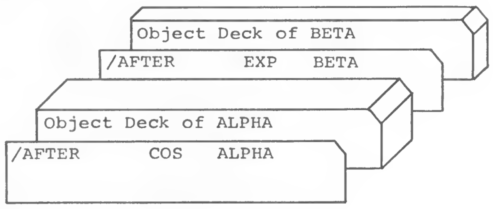
1. /AFTER must start in column 1.
2. The first blank encountered in or after column 17 results in whatever follows being regarded as comments.
3. The object subprogram deck that is to be inserted after a given library subprogram must be placed immediately following the /AFTER statement to which it corresponds. In addition, if a /AFTER statement specifies that several object subprograms are to be inserted after a given library subprogram, the object decks should appear in the same order in which their names are specified in the /AFTER statement.



Explanation: The user-written subprograms ALPHA and BETA are inserted in the system library after the subprogram COS.

4. As many /AFTER statements as needed may be used to insert subprograms in the library. However, the /AFTER statements should be arranged so that the library subprogram names are in the same order that the subprograms exist in the library.

Example: Assume that the subprogram EXP is preceded by COS in the library and that ALPHA and BETA are user-written subprograms.



Explanation: The preceding statements cause the user-written subprogram ALPHA (i.e., the object deck) to be inserted after the COS subprogram in the system library. The object deck of BETA is then inserted after the EXP subprogram. The /AFTER statements (along with the associated object decks) are ordered as shown because subprogram COS precedes EXP in the system library.

5. In an edit job, all /AFTER statements must precede any 12-2-9 EDR statements and follow any 12-2-9 REP statements.

/DELETE CONTROL STATEMENT

The /DELETE control statement is used to delete any specified library subprogram(s) from the system library.

Associated with the /DELETE control statement is a single parameter class consisting of the names of the library subprograms to be deleted.

Names of Subprograms Deleted

The names of the subprograms to be deleted must be placed starting in column 10, each separated by a comma. A complete list of all library subprogram names supplied by IBM along with the order in which they appear on the FORTRAN system tape is shown in Figure 28.

Format of /DELETE Statement

Figure 30 shows the format of the /DELETE statement.

```
Column 1                  Column 10
↑                         ↑
/DELETE                   sub₁,sub₂,...,subₙ
```

Figure 30. Format of /DELETE Statement

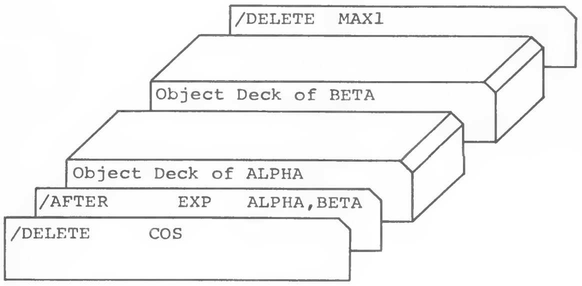
The following are considerations in using the /DELETE statements:

1. /DELETE must start in column 1.

2. The first blank encountered in or after column 10 results in whatever follows being regarded as comments.

3. /DELETE statements may be interspersed with /AFTER control statements. However, the /DELETE and /AFTER statements should be ordered so that the library subprograms to which they refer correspond to the sequence of the subprograms in the library.

   Example: Assume the subprograms COS, EXP, and MAX1 are library subprograms in that order and ALPHA and BETA are user-written subprograms.



   Explanation: The COS subprogram is deleted from the library. Because the SIN function is part of the subprogram COS, it also is deleted. The user-written subprograms ALPHA and BETA are inserted after the EXP subprogram and then MAX1 is deleted.

4. All /DELETE statements must appear after any 12-2-9 REP statements, but must precede any 12-2-9 EDR statements.

/* (ASTERISK) CONTROL STATEMENT

The /* control statement is the last statement in an edit job. It causes any remaining segments of the old system tape that were not edited to be written on the new system tape(s).

There are no parameters associated with the /* control statement. If a /EDIT and /* control statements are the only statements in an edit job, the old system tape is merely copied (written) on the new system tape(s).

The /* must appear starting in column 1. The first blank encountered after column 2 results in whatever follows being regarded as comments.

Example:

```
Column 1
↑
/*       THIS IS END OF EDIT JOB
```

12-2-9 END CONTROL STATEMENT

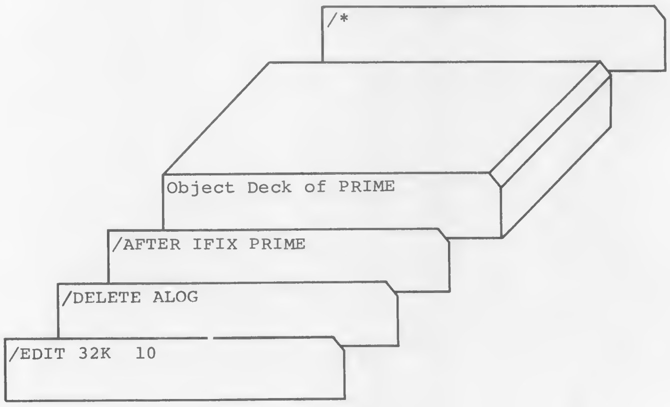
The 12-2-9 END control statement must be used to terminate an editing process for each segment specified in a 12-2-9 REP or 12-2-9 EDR control statement unless the control statements are followed by an object deck(s). In this case, the 12-2-9 END statement is not required since it is already a part of the object deck(s). The 12-2-9 END is placed starting in column 1. The first blank encountered after column 4 results in whatever follows being regarded as comments. Figure 31 shows an example of edit job using a 12-2-9 END control statement.

```
/EDIT  16K          8

  REP  002A02    /P10    C23A

  END  THIS IS END OF EDIT FOR PHASE 10

  EDR  002A46    /EDR    F20B,9E03

  END  THIS IS END OF EDIT FOR THE EDITOR
/*     REST OF TAPE COPIED ON NEW SYSTEM
```

Figure 31. Sample Edit Job

SAMPLE EDIT JOBS

Example 1: Figure 32 shows a card deck comprising a sample edit job.



●Figure 32. Sample Edit Job 1

The following is a step-by-step description of sample job 1 as it is processed:

1. After the machine operator presses the Load Key, the initial program load, FORTRAN system director, and control card routine are read into main storage. The first statement (i.e., the /EDIT statement) is read causing the editor to be read in from the system tape and main storage size to be set to 32K. Data set reference number 10 is specified as the tape on which the new system is generated. The initial program load is then written on the new system tape and the old system tape is rewound.

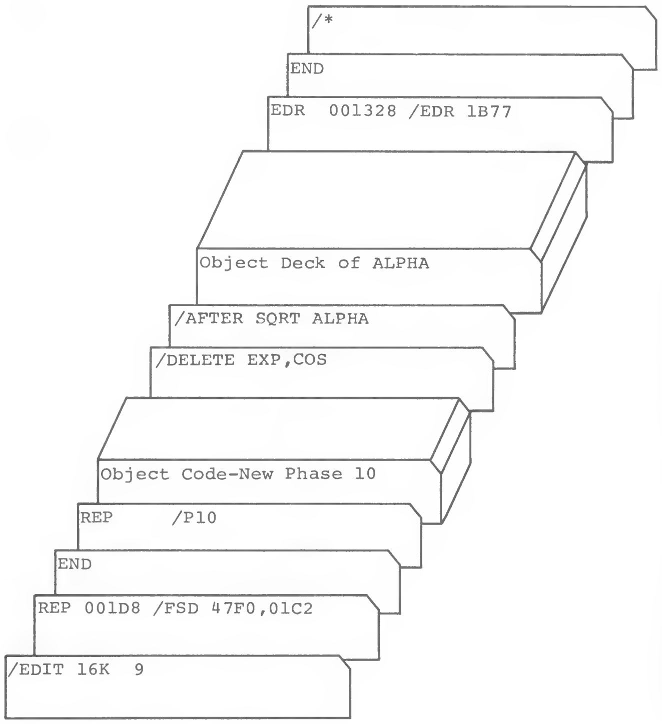
2. The next control statement (i.e., the /DELETE statement) is read in. Because it refers to the system library, all segments of the system prior to the library (i.e., FORTRAN system director, phases 10 through 30, and the relocating loader) are written on the new system tape followed by a tape mark.

3. The subprogram ALOG is then deleted from the system library.

4. The next control statement (i.e., the /AFTER statement) is read in. The object deck of the subprogram PRIME is inserted after subprogram IFIX.

5. The next control statement (i.e., the /* statement) is read in causing the remaining portion of the old system tape to be copied onto the new system tape. Both tapes are then rewound.

Example 2: Figure 33 shows a card deck comprising a sample edit job.



•Figure 33. Sample Edit Job 2

The following is a step-by-step description of sample job 2 as it is processed:

1. After the machine operator presses the Load Key, the initial program load, FORTRAN system director, and control card routine are read into main storage. The first statement (i.e., the /EDIT statement) is read causing the editor to be read in from the system tape and main storage size to be set to 16K. Data set reference number 9 is specified as the tape on which the new system is generated. The initial program load is then written on the new system tape and the old system tape is rewound.

2. The 12-2-9 REP statement causes the object code 47F001C2 to replace the existing code in the FORTRAN system director from location 0001D8 through 001DB.

3. The 12-2-9 END statement specifies that editing for the FORTRAN system director is complete. The modified FORTRAN system director is then written on the new system tape.

4. The next 12-2-9 REP statement is read in. Because it specifies that all of phase 10 is to be replaced by an object deck following the card, the control card routine (which precedes phase 10) is written on the new system tape. Then phase 10 is read from the old system tape into main storage and completely replaced by the object deck following the 12-2-9 REP statement.

5. The last statement in the object deck of phase 10 (i.e., the 12-2-9 END statement) specifies that editing for phase 10 is complete. The new phase 10 is then written on the new system tape.

6. The next control statement (i.e., the /DELETE statement) is read in. Because it refers to the system library, all segments of the compiler prior to the library (i.e., phases 12, 14, 15, 20, 25, 30 and the relocating loader) are written on the new system tape followed by a tape mark.

7. The subprograms EXP and COS are then deleted from the system library.

8. The next control statement (i.e., the /AFTER statement) is read in. The object deck of the subprogram ALPHA is inserted after subprogram SQRT.

9. The next control statement (i.e., the 12-2-9 EDR statement) is read in. Because no more modification of the library is to be made, any remaining subprograms in the library are written on the new system tape and a tape mark is written and IBCOM is copied.

10. The 12-2-9 EDR statement then causes the object code 1B77 to replace the existing code in the editor from locations 001328 through 001329.

11. The 12-2-9 END statement specifies that editing for the editor is complete. The modified editor is then written on the new system tape followed by a tape mark.

12. The next control statement (i.e., the /* statement) is read in causing the remaining portion of the old system tape to be copied onto the new system tape. Both tapes are then rewound.

This section describes the messages which signal the need for operator intervention and the procedures an operator should follow in preparing a FORTRAN job for a machine run.

The only prerequisite for setting up a FORTRAN job is a familiarity with System/360 operation.

OPERATOR MESSAGES

There are only five types of messages produced by the FORTRAN system requiring operator intervention. They are:

1. END OF JOB.
2. PAUSE xxxxx, where xxxxx is a one to five digit number.
3. Input/Output interruption message: FIS xxx, FID xxx, and FIA xxx, where xxx is the actual address of an I/O device.
4. Machine check message FMS, displayed in hexadecimal as C6D4E2 in the low-order portion of the PSW.
5. Abnormal end-of-job because of interruptions.

All messages other than the machine check message are written on the unit associated with data set reference number 15, which is usually a printer-keyboard.

END OF JOB

An END OF JOB message is printed when the job has completed its specified processing. For example, if the job is to be compiled and executed, the END OF JOB message is printed at the conclusion of the execution process.

When the END OF JOB message is printed, the operator can prepare the machine for the next job.

PAUSE

A PAUSE xxxxx message is printed when a job being executed encounters a compiled PAUSE statement.

If a PAUSE xxxxx message is printed, the operator should compare the number printed (i.e., xxxxx) with any operating instructions supplied by the programmer and perform the indicated operations. Upon completing the instructions, the operator should press the Interrupt Key to resume execution of the job.

INPUT/OUTPUT INTERRUPTION MESSAGES

There are three types of I/O interruption messages printed when an I/O device, except the 1052 Printer-Keyboard, malfunctions:

1. FIS xxx.
2. FID xxx.
3. FIA xxx.

The xxx portion of the message represents the actual machine address of the device which caused the interruption.

If the 1052 Printer-Keyboard is not functioning properly, printing is terminated.

Each message is displayed by the Instruction Counter (Storage Address) in addition to it being printed. The message is displayed in an internal representation of the external character set depending upon the hardware conversion used (e.g., EBCDIC, BCDIC, ASCII, etc.).

FIS xxx

The FIS xxx message is displayed when any one of the following conditions occur:

1. Channel Data Check.
2. Channel Control Check.
3. Interface Control Check.
4. Equipment Check.

When the FIS xxx message is displayed, the operator should address the tape (or card reader) on which SEREP (System Environment Recording and Editing Program) was placed by using the Load Address Switch and press the Load Key. Once SEREP has completed its processing, the operator must re-initialize the FORTRAN system and perform the necessary actions to rerun the error-interrupted FORTRAN job. (See the section, "Starting Instructions.")

FID xxx

The FID xxx message is displayed when any one of the following conditions occur:

1. Program Check.
2. Protection Check.
3. Command Reject.
4. Data Converted Check.
5. Data Check Read or Write.

When the FID xxx message is displayed, the operator should press the Interrupt Key to retry the I/O device that caused the malfunction. If this is ineffective, the operator should terminate job processing.

FIA xxx

The FIA xxx message is displayed when any one of the following conditions occur:

1. Data Check control.
2. Overrun by a tape device.
3. Bus Out Check.
4. Intervention required.
5. Chaining Check by a tape device.

When the FIA xxx message is displayed, the operator should make about five attempts to retry the I/O device that caused the malfunction by pressing the Interrupt Key. If this is ineffective, the operator should address the tape (or card reader) on which SEREP was placed by using the Load Address Switch and press the Load Key. Once SEREP has completed its processing, the operator must re-initialize the FORTRAN system and perform the necessary actions to rerun the error-interrupted FORTRAN job.

MACHINE CHECK MESSAGE

The hexadecimal characters C6D4E2, representing the characters FMS, are displayed in the low-order portion of the new Program Status Word (PSW) when a machine check is encountered. The entire PSW is displayed by the Instruction Counter (Storage Address) on the console.

When the characters C6D4E2 are displayed in the low-order portion of the PSW, the operator should address the tape (or card reader) on which SEREP was placed by using the Load Address Switch and press the Load Key. Once SEREP has completed its processing, the operator must re-initialize the FORTRAN system and perform the necessary actions to rerun the error-interrupted FORTRAN job. (See the section, "Starting Instructions.")

INTERRUPTION MESSAGES

When a program is interrupted and cannot recover from the exceptional condition, the job is terminated and the following message is printed:

```
ABEOJ x
```

where x indicates one of the following program exception interruption codes:

O - operation
P - privileged operation
E - execute
T - protection
A - addressing
S - specification
D - data

This message may be printed if the FORTRAN object program inadvertently alters the instructions in storage (e.g., by storing data into an array that is indexed incorrectly), or calls subprograms using incorrect arguments in the calling sequence. Execution under the GOGO option with a missing subprogram also causes object-time interrupts.

Note that the floating-point divide check interrupt and the exponent overflow and underflow cause warning messages to be written on the unit associated with data set reference number 3. These messages are described in Appendix B.

TYPES OF JOBS

There are four possible types of jobs that may be processed under control of the FORTRAN system:

1. Compile only.
2. Compile and execute.
3. Execute only (load).
4. Edit.

The type of job is specified on control cards accompanying the job. However, in the absence of control cards, the FORTRAN system assumes certain information about the job and, therefore, the operator need not add any control cards.

STARTING INSTRUCTIONS

The steps an operator should follow in preparing a machine for a FORTRAN job are:

1. Perform steps 1a, 1b, 1c, or 1d, depending on the type of job to be processed.

   a. If edit, mount at least one work tape on the device designated by the programmer. Upon completion of the job, this tape is the new FORTRAN system tape. Additional tapes may have to be mounted on tape devices, depending on the number of new system tapes to be generated by the edit job. When the new system tape(s) are generated, the operator should remove the file protection ring from the tape(s).

   b. If compile only, mount two work tapes: one on device 181, the other on 280. Go to step 2.

   c. If compile and execute, mount three work tapes: one on device 181, one on 280, and the third on 281. Device 281 (data set reference number 12) is the GO tape on which the object programs are placed. Go to step 2.

   d. If execute only (i.e., load), place the GO tape on device 281 if the object programs are to be loaded from tape. Go to step 2.

2. Mount the FORTRAN system tape on any unused device that has the same number of tracks (7 or 9) as the one that created the FORTRAN system tape. Go to step 3.

3. Using the <u>Load Address Switch</u>, address the FORTRAN system tape from the device upon which it was placed. Go to step 4.

4. Press <u>Load Key</u>.

The addresses of the actual tape devices upon which the operator places the FORTRAN system tape, work tapes, and GO tape may vary. In such cases, the operator should place the tapes on the devices specified by the programmer.

APPENDIX A: SOURCE PROGRAM DIAGNOSTICS

All diagnostic messages produced are printed in a group following the source program listing. Figure 34 shows the format of each message as it is printed under the heading of statement number, message number, and description.

When certain error conditions cannot be pinpointed to a single source statement, the error message contains an internal statement number of 0000.

Each message number (i.e., nnn) printed is preceded by the code 1FO031 as shown in Figure 34. For example, message number one is printed as:

```
S.xxxx    1FO031001   INVALID CONTROL CARD
```

The object program produced by the compilation is executed depending on:

1. The severity of the error in a particular source statement.
2. The option employed in the /JOB control statement, which determines the type of processing to be performed (i.e., GO or GOGO).

There are two types of diagnostic messages: (1) serious-error messages and (2) warning messages that an error may exist. If the GO option is specified and serious errors are encountered in the source program, execution of the object program is not attempted. If the GO option is specified and only warning messages are produced, the object program is executed. If the GOGO option is specified, object-program execution occurs irrespective of the type of errors existing in the source program.

A message whose number is immediately followed by the character "W" warns the user that an error may exist in the source statement. A serious error exists in the source statement if the message number is not followed by the character "W".

In the following text, each message number, the message itself, and its associated explanation is given. The abbreviated name preceding each explanation represents the segment of the FORTRAN system that generates the message.

The abbreviated name for each segment is:

FSD	FORTRAN System Director
CTL	Control Card Routine
P10	Phase 10
P12	Phase 12
P14	Phase 14
P15	Phase 15
P20	Phase 20
IBC	IBCOM Routine
EDR	Editor

In some cases, more than one segment could generate the same message. For example, the message:

```
S.xxxx  1FO031001    INVALID CONTROL CARD
```

could be generated by either the editor or the control card routine; therefore, the explanation is preceded by the abbreviated names EDR and CTL.

STATEMENT NUMBER	MESSAGE NUMBER	DESCRIPTION
S.xxxx	1FO031nnn	message

xxxx	is the internal statement number
nnn	is the message number
message	is the actual message printed

Figure 34. Format of Diagnostic Messages

053 SUBSCRIPT ERROR

P10 -- The subscript expression contains more than three subscripts, an illegal delimiter, or an illegal variable.

054 INVALID ARGUMENT IN ASF

P10 -- An illegal delimiter or variable appears in the statement function argument list.

055 INVALID ARGUMENT IN HEADER CARD

P10 -- An illegal variable or a multi-defined variable appears in the function definition argument list.

056 ILLEGAL STATEMENT NUMBER FIELD

P10 -- Statement number list in a computed GO TO or in an arithmetic IF statement is invalid.

057 DATA SET REFERENCE NUMBER MISSING

P10 -- There is no data set reference number specified. For example, WRITE (,10).

058 LEFT PARENTHESIS MISSING AFTER READ/WRITE

P10 -- The left parenthesis in a READ or WRITE statement is missing. For example, in the statement: WRITE 3,10), the missing left parenthesis before the 3 is needed.

060 ERROR IN VARIABLE

P10 -- Illegal variable in an EQUIVALENCE statement.

061W STATEMENT CANNOT BE REACHED

P10 -- Statement following a GO TO, RETURN, or STOP has no statement number.

063 EQUIVALENCE SUBSCRIPT ERROR

P10 -- There is an illegal delimiter or a missing subscript in an EQUIVALENCE subscript.

064 TOO MANY SYMBOLS AND STATEMENT NUMBERS

P10 -- The Dictionary Table has overflowed. Subdividing the program or reducing the number of symbols and statement numbers is necessary.

065 ILLEGAL STATEMENT NUMBER OR PAUSE NUMBER

P10 -- Either an alphabetic or illegal character is the first character in a statement number or there are more than five digits in the statement number.

066 BACKWARD DO LOOP

P10 -- The statement specified in the range of the DO statement may not precede the DO statement.

068 ERROR IN EXPONENT

P10 -- An exponent is missing or it is too big in a real or double-precision number.

069 TOO MANY ARGUMENTS IN ASF

P10 -- More than fifteen arguments in Arithmetic Statement Function definition is not permitted.

070 ILLEGAL FUNCTION NAME

P10 -- Illegal subprogram name in a FUNCTION or SUBROUTINE header statement.

071 ILLEGAL SUBROUTINE NAME

P10 -- Illegal delimiter or illegal subroutine name in a CALL statement.

072 ASF OUT OF SEQUENCE

P10 -- Arithmetic Statement Function statement is out of sequence or an array is not dimensioned prior to its first use.

073 TRANSFER TO NON-EXECUTABLE STATEMENT

P10 -- The statement number referred to by a GO TO, computed GO TO, or an arithmetic IF statement is a FORMAT or specification statement.

074 INCONSISTENT EQUATE

P10, P12 -- A variable appears in COMMON more than once or an inconsistent equate was made (e.g., the statement COMMON (A, B, C, A) is illegal).

075 UNFINISHED STATEMENT

P14 -- A right parenthesis at the end of the statement is missing.

076 PARENTHESIS ERROR

P10 and P14 -- A parenthesis is not closed or is missing. The error is substantial.

077 ILLEGAL DELIMITER OR MISSING SYMBOL

P10 and P14 -- An improper delimiter or illegal special character was encountered.

078 ILLEGAL END DO

P14 -- The last statement in the range of a DO loop cannot be a non-executable statement, Arithmetic IF, GO TO, PAUSE, RETURN, STOP, or another DO statement.

079 TYPE MUST BE INTEGER SCALAR

P10 and P14 -- The DO variable must be a non-subscripted integer variable.

080 COMMA MISSING

P14 -- A required comma was not encountered. The error is substantial.

081 ERROR IN PUNCTUATION

P10 and P14 -- Illegal decimal point or a number is missing following decimal point.

082 ILLEGAL NUMBER

P10 and P14 -- There is an error in an integer, real, or double precision number.

083 ERROR IN INTEGER

P10 and P14 -- Number zero not allowed.

084 MORE THAN FOUR WARNINGS IN STATEMENT

P10 and P14 -- Further attempt to compile the statement is terminated because four warning messages have already been generated.

085 COMPILER ERROR

P12 and P14 -- Compilers working text contains meaningless code. Job processing is continued.

086 ILLEGAL BLANK

P14 -- An illegal embedded blank was found in the FORMAT statement.

087 NUMBER MISSING

P14 -- A number is missing in E, F, T, A, I, D, or X conversion code or an illegal delimiter precedes the number.

088 NESTED PARENTHESIS

P14 -- Not more than one level of nested parentheses is permitted in a FORMAT statement.

089 ILLEGAL DATA SET REFFRENCE NUMBER

P14 -- Data set reference number must be an integer variable or constant.

090 QUOTE NOT CLOSED

P14 -- A quote mark was not found terminating the literal data in a FORMAT statement.

091 ILLEGAL SIGN

P14 -- A P format code or a blank are the only legal delimiters following a plus or minus sign in a FORMAT statement.

092 ILLEGAL COMMA

P14 -- An erroneous comma appears in the statement.

093 NUMBER TOO BIG

P14 -- The number specified in the FORMAT statement either preceding or following the format code cannot be longer than four digits.

162W BLANK CARD

P10 -- The card contains only a statement number. The card is ignored.

163W TOO MANY DIGITS IN NUMBER

P10 -- There are too many significant decimal digits in number.

164W STATEMENT NUMBER MISSING

P10 and P14 -- Format statement must have a statement number. The statement is ignored.

165W UNREFERENCED FORMAT STATEMENT

P14 -- A FORMAT statement is not referred to by any other statement. The FORMAT statement is not processed.

166W REDUNDANT COMMA

P10 and P14 -- The redundant comma in the statement is ignored.

167W LINE TOO LONG

P14 -- Format line length exceeds printer line length. Processing is completed.

168W END CARD MISSING

P10 -- The END statement is missing. Job processing continues as if it were there.

169W RIGHT PARENTHESIS MISSING

P10 and P15 -- The right parenthesis in the statement is missing. Processing continues as if it were there.

170W ZERO OR NO COUNT IN X CONVERSION

P14 -- The number preceding the X format code is 0 or blank. Processing continues.

171W PARAMETERS MISSING

P10 and P14 -- There are no parameters following a left parenthesis or a comma.

172W UNREFERENCED ASF ARGUMENT

P10 -- Argument or statement function not referred to in the arithmetic expression of an arithmetic statement function.

173W EXCESSIVE RIGHT PARENTHESIS

P10 -- The additional right parentheses in the statement are ignored and processing continues.

174W ARRAY USED AS SCALAR

P15 -- The name of the array is not followed by a subscript enclosed in parentheses.

175W STATEMENT NUMBER ON DECLARATIVE STATEMENT

P10 -- The statement number associated with the declarative statement is superfluous.

APPENDIX B: OBJECT PROGRAM DIAGNOSTICS

ERROR CODE DIAGNOSTICS

When an error condition arises during execution of a program, an error code number is produced on the unit associated with data set reference number 3. In the following text, the error codes are given in numerical order; each is followed by an explanation describing the type of error. Preceding each explanation is an abbreviated name indicating the segment of the FORTRAN system that generates the error code.

When writing out data during execution of an object program, if the internal value of a variable exceeds that allowed by the FORMAT statement, a series of asterisks (*) is written to fill the entire field.

The abbreviated name for each segment is:

IBC -- IBCOM Routine

LDR -- Relocating Loader

LIB -- FORTRAN Library

200

LDR -- A name has been omitted from the ICS card.

201

LDR -- An error has occurred in the IBCOM or reference table overlay process.

202

LDR -- An SLC, ICS, or REP statement has an invalid format.

203

LDR -- An SLC name was omitted from the SLC statement.

204

LDR -- A program to which reference was made by another program is missing.

205

LDR -- An error has occurred in the blank COMMON overlay process.

206

LDR -- An invalid ESD statement type has been detected.

207

LDR -- A duplicate symbol has been detected.

208

LDR -- The size of main storage is insufficient to process the job.

209

LDR -- An invalid loader card has been detected.

211

IBC -- An invalid character has been detected in a FORMAT statement.

212

IBC -- An attempt has been made to READ or WRITE past an end-of-data-set.

213

IBC -- The input list in a non-formatted I/O statement is larger than the logical record.

214

IBC -- An attempt has been made to WRITE more than 255 records within one logical record.

215

IBC -- An invalid character exists for the decimal input corresponding to an I,E,F or D format code.

216

IBC -- An illegal sense light number was detected in a SLITE or SLITET statement.

217

IBC -- An end-of-data-set was sensed during a READ operation.

241

LIB -- In the subprogram FIXPI(I**J), where I and J are integer variables or subscripted variables, I=0, J≤0 is illegal.

242

LIB -- In the subprogram FRXPI(R**J), where R and J are real and integer variables or subscripted variables, respectively, R=0, J≤0 is illegal.

243

LIB -- In the subprogram FDXPI(D**J), where D and J are double-precision and integer variables or subscripted variables, respectively, D=0, J≤0 is illegal.

244

LIB -- In the subprogram FRXPR(R**S), where R and S are real variables or subscripted variables, R=0, S≤0 is illegal.

245

LIB -- In the subprogram FDXPD(D**P), where D and P are double-precision variables or subscripted variables, D=0, P≤0 is illegal.

251

LIB -- In the subprogram SQRT(x), x<0 is illegal.

252

LIB -- In the subprogram EXP(x), x>174.673 is illegal.

253

LIB -- In the subprogram LOG(x) or LOG10(x), x≤0 is illegal. Because LOG10 is called by an exponentiation routine, this message also indicates that an attempt has been made to raise a negative base to a real power.

254

LIB -- In the subprogram SIN(x) or COS(x), $|x| \geq 2^{18}$ is illegal.

261

LIB -- In the subprogram DSQRT(x), x<0 is illegal.

262

LIB -- In the subprogram DEXP(x), x>174.673 is illegal.

263

LIB -- In the subprogram DLOG (x) or DLOG10(x), x≤ 0 is illegal. Because DLOG10 is called by an exponentiation routine, this message also indicates that an attempt has been made to raise a negative double-precision base to a real or double-precision power.

264

LIB -- In the subprogram DSIN(x) or DCOS(x), $|x| \geq 2^{50}$ is illegal.

PROGRAM INTERRUPT MESSAGES

Program interrupt messages containing the old Program Status Word (PSW) are produced when one of the following occurs:

1. Exponent-Overflow Exception.
2. Exponent-Underflow Exception.
3. Floating-Point-Divide Exception.

Operator intervention is not required for any of these interruptions. Figure 35 shows the interruption message format.

The three characters in the PSW (i.e., C, D, or F) represents the code number (in hexadecimal) associated with the type of interruption.

```
PROGRAM INTERRUPTION - OLD PSW IS  xxxxxxx{C|D|F}xxxxxxxx
```

Figure 35. Program Interrupt Message Format

Exponent-Overflow Exception

The exponent-overflow exception, assigned code number C, is recognized when the result of a floating-point addition, subtraction, multiplication, or division is greater than 16^{63} (approximately 10^{75}).

Exponent-Underflow Exception

The exponent-underflow exception, assigned code number D, is recognized when the result of a floating-point addition, subtraction, multiplication, or division is less than 16^{-63} (approximately 10^{-75}).

Floating-Point-Divide Exception

The floating-point-divide exception, assigned code number F, is recognized when division by a floating-point number with a zero fraction is attempted. For more information about the PSW, refer to the publication, IBM System/360 Principles of Operation, Form A22-6821.

```
AREA ---------> ┌─────────────────────────────────────────────────────┐
(word 1)        │ This word  is a part of the standard  linkage con-  │
                │ vention established under  System/360.  The space   │
                │ must be  reserved  for proper  addressing  of the   │
                │ succeeding entries.   However, an  assembled pro-   │
                │ gram may use the space for any desired purpose.     │
                │                                                     │
AREA+4 -------> ├─────────────────────────────────────────────────────┤
(word 2)        │ The address of the previous  save area;  that is,   │
                │ the save area of the  subprogram that called        │
                │ this one.                                           │
                │                                                     │
AREA+8 -------> ├─────────────────────────────────────────────────────┤
(word 3)        │ The address  of the next save area;  that is, the   │
                │ save area  of the  subprogram to which  this sub-   │
                │ program refers.                                     │
                │                                                     │
AREA+12 ------> ├─────────────────────────────────────────────────────┤
(word 4)        │ The contents of register 14  containing  the  ad-   │
                │ dress to which return is made.                      │
                │                                                     │
                │                                                     │
AREA+16 ------> ├─────────────────────────────────────────────────────┤
(word 5)        │ The contents of register 15  containing  the  ad-   │
                │ dress to which entry into this subprogram is made.  │
                │                                                     │
                │                                                     │
AREA+20 ------> ├─────────────────────────────────────────────────────┤
(word 6)        │ The contents of register 0                          │
                │                                                     │
                │                                                     │
                │                                                     │
AREA+24 ------> ├─────────────────────────────────────────────────────┤
(word 7)        │ The contents of register 1                          │
                │                                                     │
                │                                                     │
                │                                                     │
AREA+28 ------> ├─────────────────────────────────────────────────────┤
(word 8)        │ The contents of register 2                          │
                │                                                     │
                │                                                     │
                │                                                     │
                ├─────────────────────────────────────────────────────┤
                │                                                     │
      .         │                          .                          │
                ├───────────────────────         ─────────────────────┤
                │                                                     │
      .         │                          .                          │
                ├───────────────────────         ─────────────────────┤
                │                                                     │
      .         │                          .                          │
                │                                                     │
AREA+68 ------> ├─────────────────────────────────────────────────────┤
(word 18)       │ The contents of register 12                         │
                │                                                     │
                │                                                     │
                └─────────────────────────────────────────────────────┘
```

Figure 39. Save Area Layout and Word Contents

SAMPLE CALLING SUBPROGRAM LINKAGE

The linkage conventions used by an assembled subprogram which calls another subprogram are shown in Figure 40.

The coding does not have to exactly conform to that shown in Figure 40. However, the linkage should include:

1. The calling sequence by which an assembled subprogram may reference another subprogram.
2. The save and return routines by which the appropriate save area is established and control is returned to the calling program.
3. The out-of-line parameter area by which an assembled subprogram may pass parameters. (An in-line parameter area may be used instead; see the section, "In-line Parameter Area.")

LOADING OF ASSEMBLER LANGUAGE SUBPROGRAMS

The names of the library subprograms on the system tape have been modified internally to improve loading efficiency. Specifically, the program name on the ESD card of a library subprogram has been appended with a decimal point as the eighth character, with embedded blanks, if necessary, between the original name and the decimal point. The FORTRAN loader recognizes this decimal point and loads the subprogram into storage if it is called.

A subprogram written in the FORTRAN language is automatically compiled into an object program with the subprogram name as the entry point and the name ending with a decimal point as the program name. However, a subprogram written in an assembler language, which is the normal format of a user's subprogram, cannot have this facility.

The object deck may be modified manually and then edited onto the system tape to achieve faster loading.

computed by the use of the Chebyshev interpolation polynomial of degree 6 in r_0^2 in the range $0 \le r_0^2 \le 1$. The maximum relative error of this polynomial is 2^{-58}.

$\cos \frac{\pi}{4} r_0$ is computed

by using the Chebyshev interpolation polynomial of degree 7 in r_0^2 in the range $0 \le r_0^2 \le 1$. The maximum relative error of this polynominal is $2^{-64.3}$.

4. If $q \le 4$, give negative sign to the result.

The effect of an argument error is $E \sim \Delta$. As the argument gets larger, Δ grows, and since the function value is periodically diminishing, no consistent relative error control can be maintained outside the principal range $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. This holds for both sine and cosine.

CALLING SEQUENCE: Out-of-line.

DEXP Subprogram

PURPOSE: To compute the value of "e" raised to the power of a double-precision argument.

ENTRY POINT: DEXP

RANGE: $x < 174.67309$

ACCURACY: The accuracy of the DEXP subprogram is shown in Figure 58.

ARGUMENT RANGE	σ(ε) ROOT-MEAN-SQUARE Relative ERROR	M(ε) MAXIMUM Relative ERROR
$\lvert x\rvert \le 1$	7.49×10^{-17}	2.27×10^{-16}
$1<\lvert x\rvert \le 20$	8.69×10^{-16}	2.31×10^{-15}
$20<\lvert x\rvert \le 170$	9.33×10^{-16}	2.33×10^{-15}
These statistics are based on a uniformly distributed argument sample.		

Figure 58. DEXP Subprogram, Relative Error

STORAGE REQUIREMENTS: 452 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the argument is within the valid argument range; if it is not, an error message is printed and execution is terminated.

METHOD:

1. If $x < -180.2183$, give 0 as the answer.

2. Divide x by $\log_e 2$ and decompose the quotient as:

   $y = x / \log_e 2 = 4a - b - \frac{c}{16} - d$

   where a, b, and c are integers,

   $0 \le b \le 3$, $0 \le c \le 15$ and $0 \le d < \frac{1}{16}$.

   Then $e^x = 2^y = 16^a \bullet 2^{-b} \bullet 2^{-c/16} \bullet 2^{-d}$.

3. Compute 2^{-d} by using the Chebyshev interpolation polynominal of degree 6 over the range $0 \le d < \frac{1}{16}$. The maximum relative error of this polynominal is 2^{-57}.

4. If $c > 0$, multiply 2^{-d} by $2^{-c/16}$. The 15 constants $2^{-c/16}$, $1 \le c \le 15$ are included in the subroutine.

5. If $b > 0$, halve the result b - times.

6. Multiply by 16^a by adding a to the characteristic of the result.

The effect of an argument error is $\varepsilon \sim \Delta$. Since $\Delta = \varepsilon \bullet x$, for the larger value of x, even the round-off error of the argument causes substantial relative error in the result.

CALLING SEQUENCE: Out-of-line.

MOD Subprogram

PURPOSE : To compute the result of the first integer argument modulo the second integer argument. Modulo is a mathematical operator which yields the remainder function of division. Thus, 9 modulo $6 \equiv 3$.

ENTRY POINT: MOD

RANGE: Any arguments, except as noted under considerations, are acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 58 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the second argument is not zero; if it is zero, the first argument is given as the result.

METHOD: MOD (x,y)=x- [x/y] *y

The result is calculated according to the above formula, where [x/y] is the integer whose magnitude does not exceed the magnitude of x/y and whose sign is the same as x/y.

CALLING SEQUENCE: Out-of-line.

AMOD and DMOD Subprogram

PURPOSE: To compute the result of the first real-number argument (AMOD) or double-precision argument (DMOD) modulo the second argument. Modulo is a mathematical operator which yields the remainder function of division. Thus 3=39 modulo 6.

ENTRY POINTS: AMOD and DMOD.

RANGE: Any arguments, except as noted under considerations, are acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 112 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the second argument is not zero; if it is zero the first argument is given as the result.

METHOD: AMOD or DMOD (x,y) = x - [x/y] * y.

The result is calculated according to the above formula, where [x/y] is the integer whose magnitude does not exceed the magnitude of x/y and whose sign is the same as x/y.

CALLING SEQUENCE: Out-of-line.

MAX0 Subprogram

PURPOSE: To select the maximum (AMAX0 or MAX0) or the minimum (AMIN0 or MIN0) integer value from a list of integer numbers, with the option of converting the result (AMAX0 or AMIN0) to a real number or giving an integer result (MAX0 or MIN0).

ENTRY POINTS: MAX0, AMAX0, MIN0, and AMIN0

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 185 bytes.

CONSIDERATIONS: None.

METHOD: Starting with the second argument and proceeding to the end of the argument list, each argument is algebraically compared to the maximum (AMAX0 and MAX0) or to the minimum (AMIN0 and MIN0) of the previous arguments to yield a new maximum or minimum. For AMAX0 or AMIN0 the result is then converted to a real number.

CALLING SEQUENCE: Out-of-line.

MAX1 Subprogram

PURPOSE: To select the maximum (AMAX1 or MAX1) or the minimum (AMIN1 or MIN1) real value from a list of real numbers, with the option of converting the result (MAX1 or MIN1) to an integer number or giving a real-number result (AMAX1 or AMIN1).

ENTRY POINTS: MAX1, AMAX1, MIN1, and AMIN1

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 185 bytes.

CONSIDERATIONS: None.

METHOD: Starting with the second argument and proceeding to the end of the argument list, each argument is algebraically compared to the maximum (AMAX1 and MAX1) or to the minimum (AMIN1 and MIN1) of the previous arguments to yield a new maximum or minimum. For MAX1 or MIN1 the result is then converted to an integer number.

CALLING SEQUENCE: Out-of-line.

DMAX1 Subprogram

PURPOSE: To select the maximum (DMAX1) or the minimum (DMIN1) double-precision value from a list of double-precision numbers.

ENTRY POINTS: DMAX1 and DIN1

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 101 bytes.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	The number of bytes in the card. Extended card code (hexadecimal value of 0010).
13-16	Blank.
17-22	Name of entry point.
23-24	Blank.
25	Extended card code 12-1-9 (hexadecimal value of 01), identifies this as an entry point card.
26-28	The address 000008, in extended card code, of the entry point as assigned by the compiler.
29-30	Blank.
31-32	External Symbol Identification (ESID). The number 0001, in extended card code, assigned to the program in which entry points occur.
33-72	Blank.
73-76	This field is always punched with the first four characters of the program name.
77-80	This field is always punched with a sequential number of 0002.

Figure 60. ESD Type 1 Statement

ESD TYPE 2 STATEMENT

The External Symbol Dictionary (ESD) Type 2 statement points to a name within another program to which this program may refer. It is assigned an identifying number of 2 through 15, according to the order in which it is encountered among the external symbols in the program.

The format of the ESD Type 2 statement is explained in Figure 61 in terms of an 80-column card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	The number of bytes in the card. Extended card code (hexadecimal value of 0010, 0020, or 0030).
13-14	Blank.
15-16	External Symbol Identification (ESID). Sequential number, in extended card code, assigned to the first external symbol on this card.
17-22	Name of external symbol.
23-24	Blank.
25	Extended card code 12-2-9 (hexadecimal value of 02) identifies this as an external symbol card.
26-28	Extended card code 12-0-1-8-9, 12-0-1-8-9, and 12-0-1-8-9 (hexadecimal value of 000000). An address of 0 is always assigned to External Symbols by the Compiler.
29-32	Blank.
33-48	This field is used for the second external symbol and has the same format as columns 17-32. (If not used, the field is blank.)
49-64	This field is used for the third external symbol and has the same format as columns 17-32. (If not used, the field is blank.)
73-76	This field is always punched with the first four characters of the program.
77-80	This field is always punched with a sequential number.

• Figure 61. ESD Type 2 Statement

ESD TYPE 5 STATEMENT

The External Symbol Dictionary (ESD) Type 5 statement defines the existence and size of the Blank COMMON area reserved for the job.

The format of the ESD Type 5 statement is explained in Figure 62 in terms of an 80-column card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	The number of bytes in the card. Extended card code 12-0-1-8-9 and 12-11-1-8-9 (hexadecimal value of 0010).
13-14	Blank.
15-16	External Symbol Identification (ESID). The number 0002, in extended card code, assigned to the program.
17-24	Blank.
25	Extended card code 12-5-9 (hexadecimal value of 05) identifies this as a Blank COMMON card.
26-28	Address of the first byte of the program as assigned by the compiler. This field is always punched in extended card code with the hexadecimal value of 000000.
29	Blank.
30-32	Number, in extended card code, of bytes in Blank COMMON.
33-72	Blank.
73-76	This field is always punched with first four characters of the program name.
77-80	This field is always punched with a sequential number of 0003.

Figure 62. ESD Type 5 Statement

TXT STATEMENT

The Text (TXT) statement contains the instructions and/or constants of the user-written program and the starting address at which the first byte of text is to be loaded.

The format of the TXT statement is explained in Figure 63 in terms of an 80-column card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	TXT. Identifies the type of load card.
5	Blank.
6-8	24-bit starting address (in extended card code) in storage where the information from the card is to be loaded.
9-10	Blank.
11-12	Number of bytes (in extended card code) of text to be loaded from the card.
13-14	Blank.
15-16	External Symbol Identification (ESID). Number, in extended card code (hexadecimal value of 0001), assigned to the program in which the text occurs.
17-72	A maximum of 56 bytes of instructions and/or constants assembled in extended card code.
73-76	This field is always punched with the first four characters of the program name.
77-80	This field is always punched with a sequential number.

Figure 63. TXT Statement

RLD STATEMENT

The Relocation List Dictionary (RLD) statement is produced when the compiler encounters an address which is an internal

This section describes the procedures for changing the Device Assignment Table (DAT) to correspond with the I/O devices available at a particular installation.

The DAT supplied by IBM contains data set reference number assignments for a 2540 Card Read Punch with an actual address of 00C. If a user is equipped with a card read punch of which the actual address is not 00C, input cards cannot be read until the DAT is initially changed to reflect this.

The process for changing the DAT is as follows:

1. Press the Load Key. This causes the FORTRAN system director (FSD) to be loaded into storage, along with the DAT and control card routine. When control of execution is passed to the control card routine, it attempts to read from the device associated with the address 00C. If a card read punch is not associated with the address 00C, the message:

   FIS 00C

   is printed signifying an I/O error condition. The message is printed only if the physical address of the printer-keyboard is 009. Otherwise, the message will be displayed by the Instruction Counter (storage address). The system then enters a wait state.

2. Using the Load Address Switch, address the card read punch.

3. Punch the information shown in Figure 66 on an 80-column card. This card represents a program which, when executed, causes the next card in the card read punch to be read.

Column	Multiple Punch
1	12-0-1-8-9
2	12-0-1-8-9
3	12-0-1-8-9
4	12-0-1-8-9
5	12-0-1-8-9
6	12-0-1-8-9
7	0-5-9
8	12-0-1-8-9
9	12-2-9
10	12-0-1-8-9
11	12-0-1-8-9
12	12-0-8
13	11-0-1-8-9
14	12-0-1-8-9
15	12-0-1-8-9
16	12
17-80	Blank

Figure 66. Program Card Layout

4. Punch a /SET card to change the DAT on an 80-column card as shown in Figure 66.1, where 1, 3, and 15 are the data set reference numbers associated with the reader, printer, and printer-keyboard, respectively, and adr is the actual hexadecimal address of the particular device. Refer to the subsection "/SET Control Card" for details.

5. Place the program card and /SET card, in that order, in the card read punch.

```
      Column 1                     Column 10
      ↑                            ↑
      /SET                         1=adr (type), 3=adr (type), 15=adr (type)
```

•Figure 66.1. Example of /SET Card for Initial DAT Assignments

6. Press the <u>Load Key</u>. This causes the first eight columns of the program card to be placed into the Program Status Word (PSW); the second eight columns to be placed into the Channel Command Word (CCW) as shown in Figure 67.

   The instruction in the CCW causes the next card (i.e., the /SET card) in the card read punch to be read into location 88. The /SET statement causes the DAT to be changed to reflect the address of the card read punch being used.

7. After the initial change to the DAT is made, additional /SET control cards may be used to tailor the DAT to the I/O configuration of a particular installation. (See the section, "FORTRAN System Maintenance.")

```
|00|00|00|00|00|00|25|00|02|00|00|88|20|00|00|50|
<----------PSW---------><---------CCW---------->
```

Figure 67. PSW and CCW Contents After Program Card Load